Dissecting the EIP-2930 Optional Access Lists

Lioba Heimbach, Quentin Kniep, Yann Vonlanthen,
Roger Wattenhofer, and Patrick Züst

ETH Zurich
Zurich, Switzerland
{hlioba,qkniep,yvonlanthen,wattenhofer,zuestp}@ethz.ch

Abstract. Ethereum introduced *Transaction Access Lists (TALs)* in 2020 to optimize gas costs during transaction execution. In this work, we present a comprehensive analysis of TALs in Ethereum, focusing on adoption, quality, and gas savings. Analyzing a full month of mainnet data with 31,954,474 transactions, we found that only 1.46% of transactions included a TAL, even though 42.6% of transactions would have benefited from it. On average, access lists can save around 0.29% of gas costs, equivalent to approximately 3,450 ETH (roughly US$ 5 Mio) per year. However, 19.6% of TALs included by transactions contained imperfections, causing almost 11.8% of transactions to pay more gas with TAL than without. We find that these inaccuracies are caused by the unknown state at the time of the TAL computation as well as imperfect TAL computations provided by all major Ethereum clients. We thus compare the gas savings when calculating the TAL at the beginning of the block vs. calculating it on the correct state, to find that the unknown state is a major source of TAL inaccuracies. Finally, we implement an ideal TAL computation for the Erigon client to highlight the cost of these flawed implementations.

Keywords: blockchain, Ethereum, gas, transaction access list

1 Introduction

Ethereum is the second biggest blockchain in terms of market capitalization[1] and the birthplace of *decentralized finance (DeFi)*. DeFi offers many traditional financial services, e.g., exchanges and lending protocols, and has driven activity and demand for block space on the Ethereum blockchain to new levels.

On Ethereum, users pay for their transaction per unit of *gas*, i.e., a proxy for the computation cost of the transactions. Users further specify how much they are willing to pay per unit of gas. Importantly, higher paying transactions are generally prioritized. As a consequence, the total transaction fees paid by transactions on a blockchain can be seen as a proxy for the demand for block space. Ethereum users currently pay around US$ 3 Mio in transaction fees per day, overshadowing Bitcoin by a factor of three.[2]

[1] https://coinmarketcap.com (Accessed 20 September 2023)
[2] https://cryptofees.info (Accessed 20 September 2023)

Given the immense amount of money spent on Ethereum transactions, it is essential that the computational cost of transactions, i.e., the units of gas charged for the transaction, is estimated correctly. This is also necessary for preventing *denial-of-service (DOS)* attacks on the network. However, this was not always the case and as an indirect consequence, the *Transaction Access List (TAL)* emerged as a solution to contract-breaking issues arising from the needed gas increase. The TAL allows users to specify the addresses and storage keys their transaction will access. They are rewarded for providing the TAL through gas savings but are punished for including too many addresses and storage keys in their TAL. Additionally, we want to highlight that the TAL could be important to guide parallelization down the line [7, 6, 31]. For transactions with perfect TAL, it would be possible to execute transactions with no overlapping entries in parallel, which could increase Ethereum's throughput.

Contribution. In this work, we present a comprehensive study of the TAL feature from its inception on 15 April 2021 to 31 August 2023. We summarize our contributions as follows:

- We perform a longitudinal study of the usage of TALs to find that while the usage of the TAL is increasing slightly, merely 1.5% of transactions during our collection period have specified a TAL.
- For those transactions that specify a TAL we not only investigate the gas savings but also the causes of increased gas usage in 11.8% of the cases.
- Our analysis reveals that for around 71% of all transactions, which could profit from including an ideal TAL, the ideal TAL cannot be computed correctly on the state of the last block. Instead, it requires knowledge on the intra-block state the transaction executes on, which is not known ahead of time — making a proper TAL computation in these cases nearly impossible.
- Additionally, we find that all major Ethereum clients exhibit flaws in their access list computation. These imperfections can have users paying more when using the TAL computed by their client than without any.
- Finally, we implement a correct TAL computation for the Erigon client, which we made available to Erigon developers and was subsequently implemented [17]. Our implementation can also be used as a framework for the other clients.

While the TAL was a byproduct of gas increases that led to contract-breaking issues, it also has the potential to improve parallelization by providing additional information for detecting dependencies between transactions prior to execution. We demonstrate though that, due to lack of adoption and correctness, it cannot achieve this in its current state and is likely never will. The TAL feature is available for all users, who may wish to use it to achieve savings on gas fees. Thus, the challenges we present have led to and will continue to lead to unknowing users overpaying for their transactions if not addressed.

2 The History of Ethereum Transaction Costs

In Ethereum, the computational complexity of operations is measured in units of gas. Each operation in the *Ethereum Virtual Machine (EVM)* is assigned a

fixed amount of gas, representing the computational resources required for its execution. When users send a transaction to the Ethereum network, they specify the maximal gas amount a transaction is allowed to use, i.e., *gas limit*. Further, there is a minimum fee charged per transaction which is fixed per block.

This gas cost mechanism ensures that the financial cost of a transaction is proportional to the computational cost and aims to prevent DOS attacks on Ethereum. However, historical discrepancies between operation complexity and associated gas costs have led to temporary disruptions in the Ethereum network, requiring adjustments to the gas costs.

2.1 Changes to Gas Costs in Ethereum

The Ethereum Yellow Paper [34] first assigned gas costs to computational operations in 2014. Subsequently, the first adjustments were made in 2016 as part of *Ethereum Improvement Proposal (EIP)*-150 [5]. In this revamp, gas costs for IO-heavy operations were increased to better reflect their actual compute time. This adjustment was preceded by what is known as the "Shanghai Attack" [14]. Attackers found that certain Ethereum operations, i.e., `EXTCODESIZE`, `SELFDESTRUCT`, had inadequate gas costs relative to their required computation time. This led to network congestion and block creation times of more than a minute, effectively bringing the network to a halt.

In 2019, measurements revealed an imbalance between gas cost and computational complexity due to the expanding Ethereum state. The gas cost for affected operations was increased as part of EIP-1884 [32].

Then, the most significant overhaul of gas costs came in 2020 after Perez et al. [24] demonstrated that there is generally little correlation between execution cost and utilized resources. An algorithm was presented that generated transactions with a 100x higher execution time than the respective gas cost would suggest. One major problem was that state accesses cost the same independent of whether the respective addresses and storage keys were accessed for the first time in the transaction or had been accessed before (leading to faster cached accesses). EIP-2929 [8] was introduced to address this issue, distinguishing between warm and cold accesses to the Ethereum state. Naturally, the gas cost for cold accesses was set significantly higher than for warm accesses.

Year	Introduced in	Gas cost SLOAD
2014	Yellow Paper	20
2016	EIP-150	200
2019	EIP-1884	800
2020	EIP-2929	2,100 (Cold) 100 (Warm)

Table 1: Gas cost of a storage load (`SLOAD`) operation over time.

We provide an overview of the gas cost development of the `SLOAD`, i.e., loading a storage cell, instruction in Table 1. Similar changes were made to the `*CALL` and `EXT*` opcode families and the `BALANCE` and `SELFDESTRUCT` opcodes.

2.2 Transaction Access Lists in Ethereum

Together with the gas increase of EIP-2929, the TAL was proposed and later implemented in EIP-2930 [9]. This change introduced a new Ethereum transaction variant which optionally includes a parameter called "accessList", i.e., a list that

contains a set of addresses and mapped storage keys. For each entry in the TAL, an additional gas fee is charged from the sender upfront (2,400 for an address, and 1,900 for each of its storage keys). However, when an address or storage key from the list is accessed for the first time in a transaction, it is already considered warm access instead of cold access. More formally, the respective entries are added to the global sets `accessed_addresses` and `accessed_storage_keys` at the start of a transaction execution. The reduced gas costs for subsequent accesses hence compensate for the initial price to include them in the TAL (saving 100 units of gas). However, the overall gas cost of transactions may increase if non-accessed addresses and storage keys are included in the TAL.

TALs were introduced in 2020 to provide a solution to contract-breaking issues arising from the gas increase in EIP-2929. Such issues emerge when a contract calls another contract with a fixed and hard-coded gas limit. A transaction that worked before the gas cost increase could now revert because the fixed gas limit was chosen too low. By including a TAL, the gas cost of the contract call would decrease, as all accesses are considered warm; mitigating the problem.

Similarly, some contracts using default functions would not work properly anymore after the gas increase. If a contract receives Ether and no payable function is called, a fallback function is executed. This function has a gas stipend of just 2,300 units.[3] Due to the gas increase, some default functions would now run out of gas while conducting state access operations. This issue could be mitigated by including a TAL, as this would significantly decrease the gas cost of the default function by paying most of the cost for state accesses upfront.

Additionally, TALs could be used for other purposes in the future. Specifically, if all transactions included a perfect TAL, it would be possible to execute transactions with no overlapping entries in parallel. This could increase the throughput of the Ethereum network significantly.

3 Data Collection

We ran an Erigon execution layer node and a Lighthouse consensus layer node to collect Ethereum blockchain data. Our data collection window ranges from 15 April 2021 to 31 August 2023.

We take an in-depth look at August 2023 to pinpoint the achievable gas savings of all transactions with TALs and execute all transactions within this period at least six times, with (possibly) different access lists (including an empty one, the original one, and client-generated ones). To this end, we also modify an Erigon client to allow for an optimized TAL computation.

4 Analysis

We commence by studying the adoption of TALs over time (cf. Figure 1). While an initial uptrend in the adoption is visible it appears to have subsided. Further, it can be seen at no point in time was the TAL widely used.

We now go more into detail on the dataset, which spans all transactions in

[3] https://docs.soliditylang.org/en/v0.8.20/security-considerations.html#sending-and-receiving-ether (Accessed 20 September 2023)

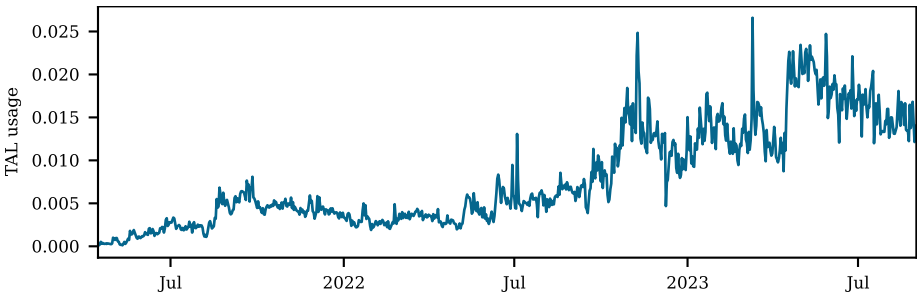


Fig. 1: Fraction of transactions per day using the EIP-2930 TALs since their introduction on April 15th 2021 (specifically, at block height 12,244,000).

August 2023 on the Ethereum mainnet. Note that over the month of August 2023, TALs were only included in roughly 1.5% of transactions. Importantly, around 20% of these TALs seen in transactions on the mainnet are suboptimal, leading to most of these transactions paying more gas fees than they would without providing a TAL. This can in part be explained by the fact that, as of this writing, none of the most popular Ethereum clients implement all the optimizations necessary to generate the most gas-efficient TALs (cf. Table 2). Common mistakes seen in the clients include the failure to adequately remove the tx sender, tx recipient, and block producer addresses which are considered warm automatically. Further, contracts created in a transaction are warm but often still included in the TAL. Finally, there is a set of warm addresses (i.e., precompiles, Ethereum addresses 0x1 through 0x9) that are considered warm as well but are still often included in TALs. Importantly, including these aforementioned addresses will lead to heightened gas costs. Hindering the adoption of the TAL even more, popular ways of interacting with the Ethereum blockchain, such as the Metamask wallet, do not support the feature at all. Finally, we highlight that it is hard to exactly predict transaction execution before inclusion in a block. The best we can reasonably expect users to do is execute the transaction on a previous state (ideally at the end of the block immediately prior). Figure 2 shows that current TAL users are often unaware of this as many transactions include too many addresses. As indicated by the shading, superfluous addresses are especially costly when they are included in the TALs.

Client	**Tx Sender**	**Tx Recipient**	**Block Producer**	**Create**	**Precompiles**
Geth			✗	✗	
Nethermind	✓	✓	✗	✓	✗
Besu	✗	✗	✗	✗	✗
Erigon		✓	✗	✗	
Ours (modified Erigon)	✓	✓	✓	✓	✓

Table 2: Possible gas optimizations for TALs, i.e., not including addresses that are considered warm (tx sender, tx recipient, block producer, created contracts and precompiles), and whether they are (as of 20 September 2023) implemented by popular Ethereum clients. Empty fields indicate that the case is handled partially. Finally, our modified Erigon client handles all cases. Note that our modifications are now implemented in the Erigon client.

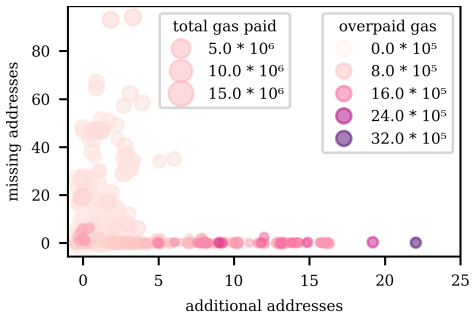


Fig. 2: Inaccuracies in original TALs.

Address type	TALs wrongfully including
Block Producer	3,710
Tx Sender	99
Tx Recipient	38,942
Precompiles	2,479

Table 3: A breakdown of the reasons for suboptimal TALs by 91,255 transactions with inaccuracies. We indicate how many TALs include the given type automatically warm of address, even though they are paying more gas fees doing so.

Frustratingly, many redundant addresses do not stem from unpredictable transaction execution, but from mistakenly adding the previously introduced commonly used addresses that do not need to be included as they are considered warm by default. Recall, that adding them into the TAL will still incur the same cost as for any other address but without an upside. Note that this is true, as long as fewer than 24 storage keys are included, in which case the benefits of adding the address and respective storage keys again outweigh the costs. Table 3 shows that a large portion of mistakes come from such entries, hinting at improper client implementations or uninformed user actions. More than a third of all suboptimal TALs incorrectly include the tx recipient in the TAL, whereas only a few include the block producer and the precompiled addresses. However, this difference might simply stem from these being accessed less frequently overall.

Looking at the profitability of different ways of generating the access lists, we realize that most of the gas savings could only be realized by knowing the exact state the transaction will be executed on (cf. Figure 3). Unless the transaction creator is also the block producer or pays for bundling, the creator has no certainty of the state on which the transaction will be executed. This has a large impact, as for around 71% of all transactions that could profit from TALs, generating the TAL at the start of the block is suboptimal.

Figure 4 visualizes the number of transactions that gain or lose gas from including TALs. Overall, we see a high concentration of transactions where TALs

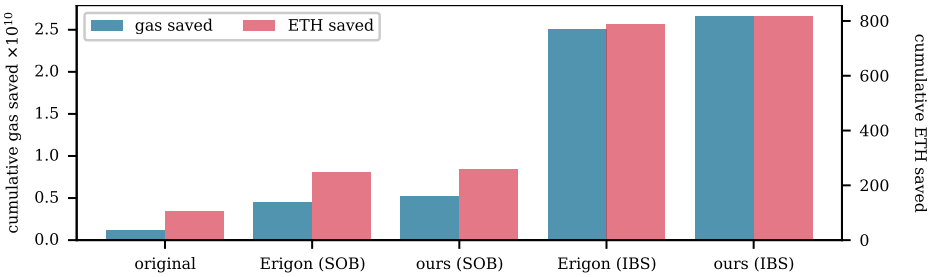
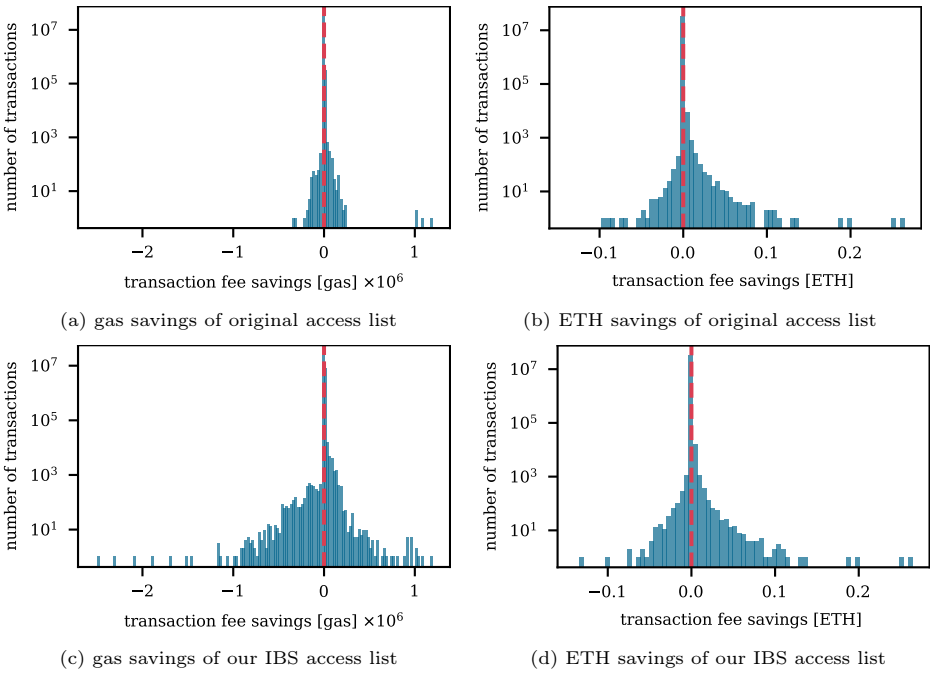


Fig. 3: Total transaction fee savings (in gas units and ETH) realized by different access lists, all compared to running the transactions without A TAL. "SOB" refers to generating TALs on the state at the start of the block, whereas "IBS" refers to generating TALs on the intra-block state.



(a) gas savings of original access list

(b) ETH savings of original access list

(c) gas savings of our IBS access list

(d) ETH savings of our IBS access list

Fig. 4: Distribution of transaction fee savings (in gas units and ETH) realized by the original TALs as well as our optimized TALs generated on the intra-block state (IBS).

have little impact. In Figure 4d we plot the effect of adding optimal TALs. Note that while upsides are increased, adding a (more precise) TAL can lead to higher gas usage in rare cases, i.e., 0.6% of all transactions in our analysis. However, this can be explained by transaction behavior changing, e.g., due to parts of the transaction running out of gas later. Nevertheless, in reality, current users seem to be more conservative in their choice of TAL as shown by Figure 4a.

Finally, we observe that original access lists disproportionately realize gas savings. They reach around a fourth of total potential gas savings (and even over 40% when measured in ETH), compared to calculating access lists for all transactions at the start of each block, even though these represent just 1.5% of transactions. This underlines the claim that currently TALs are often used by rather well-versed users on large transactions, that are able to capture a considerable part of benefits that are to be had.

5 Related Work

Gas Prices. Given the immense total cost of Ethereum transactions, an active line of research is devoted to Ethereum gas. One important aspect of Ethereum gas is predicting the next block's price of gas as it is essential for quick block inclusion at fair prices. Multiple works are invested into predicting the price of gas [33, 23, 19, 25, 26, 10], as well as understanding the impact on sudden price surges [12].

Given the negative externalities of unpredictable gas prices, EIP-1559 was introduced in 2019. EIP-1559 was shown to be incentive compatible and to sta-

bilize gas prices [28, 18]. Multiple empirical works have further demonstrated the promised improvements in Ethereum transaction fees [27, 21]. Still, it has been shown that in the presence of farsighted miners/validators [4, 16] rational attacks on the transaction fee mechanism can lead to less predictable fees.

As opposed to these works, our work does not concentrate on gas prices but on the amount of gas that can be saved by users with the TAL.

Gas Usage. Multiple works investigate the usage of gas on Ethereum [22, 1, 30, 19, 35], e.g., how much gas is used by smart contracts and whether there are possible optimizations to reduce the gas usage. Our work is orthogonal to these, as we focus on the possible gas savings using transaction access lists.

A separate line of work investigates the Ethereum gas exceptions [2, 3, 20, 13]. The TAL, which we investigate, is a byproduct of gas exceptions in various smart contracts, but its impacts were hoped to extend far beyond.

Perez et al. [24] examine the utilized resources of Ethereum transactions to demonstrate that there is generally little correlation between execution cost and utilized resources. As a result of their work, EIP-2929 and TALs were created. To the best of our knowledge, our study is the first to analyze TAL usage and future potential.

Parallelization. Saraph and Herlihy [29] and Chen et al. [11] empirically estimate the potential concurrency of speculative execution in Ethereum, while Heimbach et al. [15] quantify the concurrency limits of Ethereum workload assuming accesses are known in advance. In contrast, our work explores to what extent advance knowledge of accesses is realistic.

6 Concluding Discussion

Overall we saw that the potential gas savings are significant when considered in absolute terms. However, incentives for individual users are small and currently seem insufficient for most users to put in the necessary effort to make proper use of this feature. Even more so, (uninformed) users risk overpaying when including a TAL as a result of flawed TAL computations by all major Ethereum clients and inherent difficulties in proper TAL computations given that the state the transaction executes on is not known ahead of time. A first remedy could arrive from smart contract developers themselves. By providing clear instructions on optimal TAL construction for their developed smart contracts, their users could avoid potential losses, possibly resulting in mutual benefits.

For now, though, the current lack of adoption also makes TALs virtually unusable for parallelizing transactions. Performing parallel scheduling with (partial) knowledge of dependencies between just 1.5% of all transactions would have a negligible impact on the overall performance of the network.

Driving the adoption of TALs could be done through mechanism design, for instance by increasing the potential gas savings if a TAL is provided by the user. However, getting the incentives for TALs right could prove to be a difficult task. On the one hand, Ethereum's goal of making gas cost more correlated with computation time would call for incentivizing accesses to warm (cached) data. Yet, to increase the parallelizability of transactions the opposite seems advisable: disincentivizing accesses to frequently used state (at least across transactions).

References

1. Albert, E., Correas, J., Gordillo, P., Román-Díez, G., Rubio, A.: Gasol: Gas analysis and optimization for ethereum smart contracts. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 118–125. Springer (2020)
2. Albert, E., Gordillo, P., Rubio, A., Sergey, I.: Running on fumes: Preventing out-of-gas vulnerabilities in ethereum smart contracts using static resource analysis. In: Verification and Evaluation of Computer and Communication Systems: 13th International Conference, VECoS 2019, Porto, Portugal, October 9, 2019, Proceedings 13. pp. 63–78. Springer (2019)
3. Ashraf, I., Ma, X., Jiang, B., Chan, W.K.: Gasfuzzer: Fuzzing ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities. IEEE Access **8**, 99552–99564 (2020)
4. Azouvi, S., Goren, G., Hicks, A., Heimbach, L.: Base fee manipulation in ethereum's eip-1559 transaction fee mechanism. arXiv preprint arXiv:2304.11478 (2023)
5. Buterin, V.: EIP-150: Gas cost changes for io-heavy operations. `https://eips.ethereum.org/EIPS/eip-150` (September 2016), Ethereum Improvement Proposals, no. 150
6. Buterin, V.: Easy parallelizability. `https://github.com/ethereum/EIPs/issues/648` (June 2017)
7. Buterin, V.: The stateless client concept. `https://ethresear.ch/t/the-stateless-client-concept/172` (October 2017)
8. Buterin, V., Swende, M.H.: EIP-2929: Gas cost increases for state access opcodes. `https://eips.ethereum.org/EIPS/eip-2929` (September 2020), Ethereum Improvement Proposals, no. 2929
9. Buterin, V., Swende, M.H.: EIP-2930: Optional access lists. `https://eips.ethereum.org/EIPS/eip-2930` (August 2020), Ethereum Improvement Proposals, no. 2930
10. Carl, D., Ewerhart, C.: Ethereum gas price statistics. University of Zurich, Department of Economics, Working Paper (373) (2020)
11. Chen, Y., Guo, Z., Li, R., Chen, S., Zhou, L., Zhou, Y., Zhang, X.: Forerunner: Constraint-based speculative transaction execution for ethereum. In: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. pp. 570–587 (2021)
12. Faqir-Rhazoui, Y., Ariza-Garzón, M.J., Arroyo, J., Hassan, S.: Effect of the gas price surges on user activity in the daos of the ethereum blockchain. In: Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems. pp. 1–7 (2021)
13. Grech, N., Kong, M., Jurisevic, A., Brent, L., Scholz, B., Smaragdakis, Y.: Madmax: Surviving out-of-gas conditions in ethereum smart contracts. Proceedings of the ACM on Programming Languages **2**(OOPSLA), 1–27 (2018)
14. Greene, R., Johnstone, M.N.: An investigation into a denial of service attack on an ethereum network (2018)
15. Heimbach, L., Kniep, Q., Vonlanthen, Y., Wattenhofer, R.: Defi and nfts hinder blockchain scalability. In: Financial Cryptography and Data Security (FC), Bol, Brac, Croatia (May 2023)
16. Hougaard, J.L., Pourpouneh, M.: Farsighted miners under transaction fee mechanism eip1559. In: 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 1–9. IEEE (2023)

17. Kniep, Q.: Extend gas optimization for eth_createaccesslist. https://github.com/ledgerwatch/erigon/pull/8261 (September 2023)
18. Leonardos, S., Monnot, B., Reijsbergen, D., Skoulakis, E., Piliouras, G.: Dynamical analysis of the eip-1559 ethereum fee market. In: Proceedings of the 3rd ACM Conference on Advances in Financial Technologies. pp. 114–126 (2021)
19. Li, C.: Gas estimation and optimization for smart contracts on ethereum. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 1082–1086. IEEE (2021)
20. Liu, C., Gao, J., Li, Y., Chen, Z.: Understanding out of gas exceptions on ethereum. In: Blockchain and Trustworthy Systems: First International Conference, BlockSys 2019, Guangzhou, China, December 7–8, 2019, Proceedings 1. pp. 505–519. Springer (2020)
21. Liu, Y., Lu, Y., Nayak, K., Zhang, F., Zhang, L., Zhao, Y.: Empirical analysis of eip-1559: Transaction fees, waiting times, and consensus security. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2099–2113 (2022)
22. Marchesi, L., Marchesi, M., Destefanis, G., Barabino, G., Tigano, D.: Design patterns for gas optimization in ethereum. In: 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE). pp. 9–15. IEEE (2020)
23. Mars, R., Abid, A., Cheikhrouhou, S., Kallel, S.: A machine learning approach for gas price prediction in ethereum blockchain. In: 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC). pp. 156–165. IEEE (2021)
24. Perez, D., Livshits, B.: Broken metre: Attacking resource metering in EVM. In: 27th Annual Network and Distributed System Security Symposium. https://www.ndss-symposium.org/ndss-paper/broken-metre-attackingresource-metering-in-evm
25. Pierro, G.A., Rocha, H.: The influence factors on ethereum transaction fees. In: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). pp. 24–31. IEEE (2019)
26. Pierro, G.A., Rocha, H., Tonelli, R., Ducasse, S.: Are the gas prices oracle reliable? a case study using the ethgasstation. In: 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE). pp. 1–8. IEEE (2020)
27. Reijsbergen, D., Sridhar, S., Monnot, B., Leonardos, S., Skoulakis, S., Piliouras, G.: Transaction fees on a honeymoon: Ethereum's eip-1559 one month later. In: 2021 IEEE International Conference on Blockchain (Blockchain). pp. 196–204. IEEE (2021)
28. Roughgarden, T.: Transaction fee mechanism design for the ethereum blockchain: An economic analysis of eip-1559. arXiv preprint arXiv:2012.00854 (2020)
29. Saraph, V., Herlihy, M.: An empirical study of speculative concurrency in ethereum smart contracts. arXiv preprint arXiv:1901.01376 (2019)
30. Signer, C.: Gas cost analysis for ethereum smart contracts. Master's thesis, ETH Zurich, Department of Computer Science (2018)
31. Sun, X.: Speeding up the ethereum virtual machine (part 1). https://collective.flashbots.net/t/speeding-up-the-evm-part-1/883 (January 2022)
32. Swende, M.H.: EIP-1884: Repricing for trie-size-dependent opcodes. https://eips.ethereum.org/EIPS/eip-1884 (March 2019), Ethereum Improvement Proposals, no. 1884
33. Werner, S.M., Pritz, P.J., Perez, D.: Step on the gas? a better approach for recommending the ethereum gas price. In: Mathematical Research for Blockchain

Economy: 2nd International Conference MARBLE 2020, Vilamoura, Portugal. pp. 161–177. Springer (2020)
34. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)
35. Zarir, A.A., Oliva, G.A., Jiang, Z.M., Hassan, A.E.: Developing cost-effective blockchain-powered applications: A case study of the gas usage of smart contract transactions in the ethereum blockchain platform. ACM Transactions on Software Engineering and Methodology (TOSEM) **30**(3), 1–38 (2021)